

PARALLEL COMPUTATION IS ESS

NABARUN MONDAL AND PARTHA P. GHOSH

ABSTRACT. There are enormous amount of examples of Computation in nature, exemplified across multiple species in biology. One crucial aim for these computations across all life forms are their ability to learn and thereby increase the chance of their survival. In the current paper a formal definition of autonomous learning is proposed. From that definition we establish a General Purpose model for learning. Different implementations of the model are discussed. It is found that for general purpose learning, the models capable of parallel execution would be evolutionarily stable, and hence in Nature parallelism in computation is found in abundance.

1. INTRODUCTION

Computations are abound in nature, and it is not hard to fathom that one of the purposes of the natural computation is to learn. A more learned individual would thereby gather advantage over the others [17], and survive. But then, how the individuals of a species learn? What is the mechanism behind their learning?

The extended Church Turing Thesis [14][15] [2][3][4] normally stated in the form: *any computation happening in nature must also have a Turing Machine analog*; suggests that no computation in nature can exceed the power of a Turing Machine [6]. This intriguing thought of simulating nature on Turing Machines provoked the study of machine learning and artificial intelligence [7] [5].

One school of thought about machine learning follows the sequential way of creating machines which are capable of learning. In fact scholars from the sequential school argue [14][15] that parallel systems won't have any added advantage over the sequential ones, because both are reducible to Turing Machines.

Another school of thought [1] questions the sequential learning strategies, as nature seems to be inherently running in parallel. The massive parallel structures

2010 *Mathematics Subject Classification.* Primary 68Q32 ; 68Q05 ; Secondary 68T05 ; 68Q10 ; 62P10 ; 97M60 ; 92D15 ; 91A80 .

Key words and phrases. learning ; Turing Machines ; Models of Learning ; Decidability ; sequential ; parallel computation ; clone ; mutation ; evolution ; Nash Equilibrium ; Evolutionary Stable Strategy .

Dedicated to my late professor Dr. Prashanta Kumar Nandi.

Dedicated to my parents without their presence we are nothing.

Big thanks to: Abhishek Chanda, Gurpreet Singh (Manager), Waseem Fazal, Sahiba Khurana, Jaipal Reddy : You all have been constant support.

In Memory of : Dhruvajyoti Ghosh, my closest human being. RIP. .

of neurones in brain [13] prompted the artificial neural network studies [9] [10], and it is now known that bacterial colony learn by pooling in their individual (bacterium) resources [12] which makes the learning strategy parallel by definition. Scholars from the *parallel* school ask the question :-

Given Nature is inherently parallel, how useful the sequential way of learning going to be?

It is well known that computationally both of these models actually yield the same power [4] [8] : *every parallel system must have a sequential analog*. The sequential models would take more time, less computing power, and less space, while the parallel ones would take less time, more computing power and more space. Clearly then, the solution to Natures parallelism lies not in the computing power of the abstract Turing Machine, but had to do with how each model has evolved in nature with no supervision (blind design) [16].

Can one then decisively argue the need for which parallel computation is prevailing in Nature? This question begs an answer because parallelism in execution is harder to attain in the artificial settings of man made computing machines. Artificial parallel systems are faster, but too simple when compared against the natural systems like human brain, which are slower but far more complex[14].

In the present paper we take up this exact problem. We notice that the real problem about systems evolving in nature is:-

Given limited economy of operations possible in nature, what mode of computation would evolutionarily arise and would become dominant?

We discuss the abstract learning procedure in the Section 2. We establish that there is an abstract autonomous (blind,directionless) system capable of learning, which borrows two operations from biology : cloning and mutation. Section 3 discusses sequential and parallel design (Dawkins *designoid*, as any evolved, blind design is not a design in a true sense [17]). In the Section 4 we compare these two models from optimality or resources standpoint. We find that for general purpose learning, parallelism has evolutionary advantage. It is not only because it is faster, but also because using the operations *clone* and *mutate* no other *computationally* better organization is possible, and *computationally* it can not be improved. Finally in the Section 5 we put all these results together and show that once parallel strategy (in the sense of game theory) has invaded the population, it would become dominant, and stay dominant, because it is an *evolutionarily stable startegy*. Therefore, we conclude that, the theory of computation and game theory together can explain the prevalence of parallelism in computational circuitry in nature.

2. LEARNING

Everyone has an intuitive idea about what learning is. But that idea relies upon another intuitive idea of what is knowledge. For example *Learning is, knowing what one did not know earlier*.

In this sense, there are some properties of learning one can summaries:-

Definition 2.1. Informal Description of Learning.

After a system has learned, at some time t_L the following properties hold:-

- (1) *The system could achieve something which was unachievable at t , $t < t_L$.*
- (2) *The system would continue achieving everything it could achieve before t_L .*
- (3) *The system should be autonomous, devoid of supervision by any intelligent agent.*

The third point needs elaboration. Once the system is set into motion, after that no tweaking should be done with the system. In specificity *getting out of the system to perform system optimization* is not allowed at all for autonomous learning. This jumping out of the system idea is elaborated heavily in [11].

In a formal sense then, if there is a set of *knowledge* associated with the system which can somehow be identified as the set $\mathcal{K}(t_1)$ at time t_1 , and at a later time t_2 with the set $\mathcal{K}(t_2)$, then the statement *a learning took place between t_1 to t_2* is equivalent to the formal statement:-

$$\mathcal{K}(t_1) \subset \mathcal{K}(t_2).$$

The learned knowledge can be modelled by :-

$$\mathcal{L}(t_1, t_2) = \mathcal{K}(t_2) \setminus \mathcal{K}(t_1)$$

But these semi-formal definitions would not really formalize learning given that the set $\mathcal{K}(t)$ was not formally defined. Can we formally quantify the set $\mathcal{K}(t)$?

At this point, we argue that the existence of $\mathcal{K}(t)$ can not be measured without the effect of $\mathcal{K}(t)$ exhibited by the system, which should only be identified by experimentation. If the *Extended Church Turing thesis* [14][15][4] is true, that is any model of computation has an analogous Turing Machine model, then, the effect of $\mathcal{K}(t)$ can be found in pretty straight forward manner. We would need some definitions to reach to that formal definition of learning.

Definition 2.2. System.

Let a system S be defined by an underlying Universal Turing Machine \mathcal{TM} [4] with a rule table $R(S)$ which accepts the language class $C(S)$, by simulating the rule table $R(S)$.

Definition 2.3. Formal Definition of Learning.

Let a system (definition 2.2) at time t accepts the language class $C(S(t))$. The system has learned within time $t_1 \rightarrow t_2, t_1 \leq t_2$ iff:-

$$C(S(t_1)) \subset C(S(t_2))$$

The knowledge of the system $\mathcal{K}(S) = C(S)$, and learning is precisely defined as:-

$$\mathcal{L}(t_1, t_2) = C(S(t_2)) \setminus C(S(t_1))$$

Comparing the definitions (2.1) with (2.3) we can see that (2.3) is just a type of (2.1) which matches with the intuition. Knowledge accumulation is now experimentally verifiable. To show that a system has learned between (t_1, t_2) , one has to find a string $w_2 \notin C(S(t_1))$, but $w_2 \in C(S(t_2))$. We also note that the strict subset inclusion $C(S(t_i)) \subset C(S(t_j))$ for $i < j$ makes learning a *filtration* [19] [20] process.

We note that the language class $C(S)$ accepted by a system S relies upon the rule table $R(S)$ to be used by the underlying Universal Turing Machine, and nothing else. Hence, learning process must *somehow* alter the rule table.

But what is the *natural* mechanism? How an altered rule table ensures the earlier accepted language class still remains to be accepted even after the modifications of the rule table? Also, what kind of alternations to the rule table are permitted? From a biological point of view, a mechanism seems feasible.

In Biology, the change(improvement) of a life form takes place due to a change in the biological rule book, that is, the genome [16]. It has been argued [16] [17] that the rule book is the only thing that matters, and identifies the life form. The change in the life form, no matter how complex it looks like propagates in small, random changes in the genome. These changes have very small but non zero probability [17], and can be modelled as a stochastic process [19] [20]. *Two different life forms having same genetic code are biologically indistinguishable*. Interestingly, the same argument holds true here. For a system, the rule table is the only thing that matters, there is no way to distinguish two different systems knowledge, given that they are having the same copy of the rule table!

The process of copying the rule book is known as *cloning*, while any modification is known as *mutation* which generates a different rule book. We now define these concepts in a more formal way.

Definition 2.4. Clone System.

A system S_c is said to be clone of another system S iff the rule table of both the systems are identical. Formally:-

$$R(S_c) = R(S)$$

Given a system S capable of making a clone S_c of itself, then, it can ensure that $C(S) = C(S_c)$, that is both system would accept the same language class, thereby exhibiting same knowledge, as discussed before.

Now, we need to discuss about how to alter the rule table information. To do that we borrow a groundbreaking idea from Gödel.

Definition 2.5. Gödel Encoding (Gödelization).

Let the alphabet $\Sigma = \{s_0, s_1, s_2, \dots, s_{b-1}\}$ with $|\Sigma| = b$. Let $g : \Sigma \rightarrow \{0, 1, 2, 3, \dots, b\}$ and be defined as:-

$$g(x) = i \text{ when } x = s_i$$

Then, the Gödelization of the string $w = x_0x_1...x_k$ is defined as:-

$$G(w) = \sum_{r=0}^k g(x_r)b^r$$

Definition 2.6. Gödel Decoding (Reverse Gödelization).

Let the alphabet $\Sigma = \{s_0, s_1, s_2, \dots, s_{b-1}\}$. with $|\Sigma| = b$. The reverse Gödelization of x is defined as inverse function of Gödelization i.e. $\rho_g : \mathbb{Z}_+ \rightarrow \Sigma^+ :-$

$$\rho_g(x) = w \text{ iff } G(w) = x$$

To learn, we need a mechanism to change the rule table. In biology, such a mechanism exists, and is known as *mutation*. We use the same keyword, and define mutation formally.

Definition 2.7. Mutation. Let w is a string from alphabet Σ . Let $r_c(x)$ be a class of computable functions $r_c : \mathbb{Z}_+ \rightarrow \mathbb{Z}_+$ such that:-

$$\forall x \in \mathbb{Z}_+ ; r_c(x) \neq x$$

Then, the function $\mu : \Sigma^+ \rightarrow \Sigma^+$

$$\mu(w) = \rho_g(r_c(G(w)))$$

is called a mutation of the string w .

Definition 2.8. Mutation of A System.

Let the Gödelization of the rule table R of a system S be defined as:-

$$R_\mu = \mu(R)$$

A system S_μ having $R(S_\mu) = R_\mu$ as a rule table is called mutated version of the system S .

The notion of mutation of a system (definition 2.8) explains the idea how new rules can get created automatically. However, the old rules should not get deleted. These principles are explain in the axioms which follow next.

Axiom 2.1. Natural Learning Systems Axioms.

- (1) Learning is as defined as definition 2.3.
- (2) Only allowed operations on rule tables are Clone and Mutation.

We now show that a model exists which is capable of autonomous learning as in definition 2.3. It adhere to the axioms of (2.1) and learns without any help from any supervisor. Simply speaking we show that a designoid [17] can learn.

Theorem 2.1. There is a learning process following Axiom (2.1).

Let $S(t_1)$ be a system at t_1 with a non deterministic Universal Turing Machine. Let any point of time t_2 , S cloned itself, (definition 2.4) and then muted (definition

2.7) the clone system to generate a mutated system (definition 2.8). At t_2 therefore there are two systems :-

$$S(t_2) = (S_c, S_\mu)$$

The non deterministic universal Turing Machine of S can non deterministically use rule table R_c or R_μ . Given that $C_N = C(S) \setminus C(S_\mu) \neq \emptyset$, this system has learned the language class C_N within the interval (t_1, t_2) .

Proof of the theorem 2.1 . We have $C_N \neq \emptyset$. Given a string $w_s \in C(S)$ and $w_s \notin C(S_\mu)$, by construction, this string would be accepted by the new system given the rule table R_μ is used non deterministically.

In the same way a string $w_n \in C_N$ such that $w_n \notin C(S)$ would be accepted by the new system given the rule table $R(S)$ is used.

Therefore, the new system would continue accepting any string that was earlier acceptable, and also would accept new strings which it did not accept earlier. Therefore, the new system qualifies as a system which has learned. \square

3. DIFFERENT MODELS OF LEARNER

We have established that there exists a system depicted in Theorem 2.1 which is capable of learning by mutation and cloning. The systems which are capable of learning subsequently would be called *learners*.

It is easy to notice that by the Theorem 2.1, the way a learner really learns is adding a new rule table to its repository of existing rule tables. For example if the original rule system was R_1 , then the learned system has two rule tables $R_1, \mu(R_1)$ which can be said as $\{R_1, R_2\}$. In the next level of learning, the rule tables would become:-

$$\{R_1, R_2, \mu\{R_1, R_2\}\} = \{R_1, R_2, R_3\}$$

So, at every steps of learning, formally one new rule table gets added, and the learner conveniently must switch from one table to another to accept a string. *Note that no internal shuffling of the rules are allowed, the rule tables are atomic building blocks by axiom (2.1).* With this idea in mind we can now formally define a learner as follows:-

Definition 3.1. A Learning System : Learner.

A learner is a system S comprise of a set of rule tables $\mathcal{L}(S) = \{R_k\}$, and a Universal Turing Machine. By simulating a rule table R_k it can accept strings $w_k \in C_k \subset C(R_k)$. It is capable of adding new rule table to $\mathcal{L}(S)$ via a process called learning. To learn the system can add a mutated copy of any of the R_k to $\mathcal{L}(S)$:-

$$\mathcal{L}(S)_{t_2} = \mathcal{L}(S)_{t_1} \cup \{\mu(R_k)\}$$

The language class accepted by the learner is :-

$$C(S) \subseteq \bigcup_k C(R_k)$$

Definition 3.2. Complete Learner.

Let's assume a learner S has a set of rule tables as $\mathcal{L}(\mathcal{S}) = \{R_k\}$. Let the individual language class for each R_k be $C(R_k)$. Let the language class accepted by the learner be C_L . The learner S is said to be complete iff:-

$$C_L = \bigcup_k C(R_k)$$

As for mechanism of implementation of a learner, it can belong to two general classes *tandem* or *sequential* class, and *parallel* class.

Definition 3.3. Sequential Learner.

To accept a string a sequential learner sequentially picks one rule table R_k from $\mathcal{L}(\mathcal{S})$ and simulates it using the Universal Turing Machine, until either no unused rule table exists, or there is an accept.

Which rule table to be used after which rule table becomes of importance now. There is no obvious answer to that. In fact this question, the inherent sequential nature limits the language class the sequential system can accept. This is the subject matter of the next theorem.

Theorem 3.1. Language Class Accepted by the Sequential Learner.

Sequential learner accepts a language class C_S which is strictly a subset of the union of the language class of all the rule tables R_k s.

$$C_S \subset \bigcup_k C(R_k)$$

Proof. We construct the exact language class accepted by the sequential learner. We begin by constructing the class of strings which would be accepted by the learner. Let $C_u(k)$ be the set of strings those are accepted by rule table R_k , and , gets either accepted or rejected by any $R_r \in \mathcal{L}(\mathcal{S}) \setminus \{R_k\}$, that is all string from the class $C_u(k)$ makes the underlying Universal Turing Machine halt at all the other rules, but would not get into any infinite loop.

Then, the language class accepted by the sequential learner is precisely :-

$$C_S = \bigcup_k C_u(k)$$

It is obvious that $C_u(k) \subseteq C(R_k)$, and therefore, in general, when there exists at least a string in rule table R_k which makes the underlying Universal Turing machine simulating on rule table R_j get into infinite loop, we would have:-

$$C_S \subset \bigcup_k C(R_k)$$

That completes the proof. □

It can be argued however that by *cleverly* ordering the selection of the rule tables one can possibly complete (definition 3.2) the learner. But in general that will be impossible. Next theorem establishes this fact.

Theorem 3.2. *Sequential learner can not be completed.*

Algorithmic modification of a sequential learner into completeness is impossible.

Proof. We demonstrate using the worst case scenario, which is $\forall R_k \exists w_k$ such that simulating R_j with input w_k gets the universal turing machine into infinite loop.

Obviously, it is impossible to reduce such a system into a complete system. Now, assume that there is only one $w_k \in R_k$ such that simulating R_h with input w_k gets the universal turing machine into infinite loop.

But that is unknown unless we are looking at the system from outside of the system. We must then already have a table which shows what strings from which rule table class produce a infinite loop in which rule table.

The question is can we create such a table algorithmically? The answer is no, because we would never know which string would gets the simulation into infinite loop. Therefore, automatic creation of such a table is not possible.

Hence, automatic completion of the sequential system, is not possible either, as was stated. \square

Therefore, it is now established by the Theorem 3.1 that a sequential learner does not have closure of the language class for which it poses the rule tables never the less. Also, it is not possible to automatically complete it, as stated by Theorem 3.2.

Although the rules in the rule table lets it precisely accept the language class C_x it might still not accept all the strings from the class. This is obviously not efficient.

The next learner, does not have that inefficiency, but it achieves that goal with extra processing units, and space. The next learner, of course is the parallel learner. It utilizes the concept of parallel running Turing Machines [8] having dedicated tapes each, but communicate with a monitoring Turing Machine using a shared tape.

Definition 3.4. *Parallel Learner.*

Let $n = |\mathcal{L}(\mathcal{S})|$. Then the parallel learner has $n + 1$ Universal Turing Machines. n of them are having single dedicated tape, and a shared tape which everyone shares.

To accept an input string $w \in \Sigma^+$ w is to be supplied to the shared tape, from where all the Turing Machines copy the string into their own dedicated tape, and start running the simulation. If one of them \mathcal{TM}_a could accept the string, \mathcal{TM}_a writes back a symbol $\mathcal{V} \notin \Sigma$ to the shared tape.

The last Turing Machine only monitors the shared tape for the symbol \mathcal{V} . If it has found the symbol, it accepts the string, as the system has accepted it.

Theorem 3.3. *Language Class Accepted by the Parallel Learner.*

Parallel learner accepts a language class C_P which is strictly the union of the language class of the all the rule tables R_k s.

$$C_P = \bigcup_k C(R_k)$$

Proof. This is elementary. The problem of a simulated system getting into an infinite loop is solved by the n Turing Machine running parallel without affecting each others progress. Also the last Turing machine monitoring any of the simulation output ensures if any Turing machine halts with accept, the learner would halt. Therefore, the parallel system is strictly complete over the rules. \square

4. COMPARISONS OF LEARNING STRATEGIES

In the previous section, we have established two learning strategies, one in tandem, or the sequential strategy (definition 3.3), another - the parallel strategy (definition 3.4). In the current section we establish the pros and cons of the different strategies.

Note that we are discussing a *blind* design, that is no conscious optimization, ever. None is looking at the system from the outside, and improving the design or the wiring of the system.

Theorem 4.1. *Completeness of the Learning Models.*

Parallel Learners are complete (definition 3.2) while Sequential Learners are not.

Proof. The proof is a direct consequences of theorem 3.3 and theorem 3.1. \square

By this theorem, one clear thing we have established is that the parallel system is more *optimal* than the sequential system, in general. That is, a parallel system can use all it's resources to gain maximum coverage on the strings those are to be accepted, while a sequential learner can not. Also by Theorem 3.2 we have established that there is no autonomous way to complete the sequential system. So, parallel systems has inherent evolutionary advantage.

But, this is not the only metric in which the learners to be measured for optimality. In computer science, the time and space complexity are of utmost importance.

Formally the time complexity question is *how much time it takes to accept a string?*

It is not too hard to show that the parallel strategy wins here. We show it in the next theorem:-

Theorem 4.2. *Time Complexity Comparison of the Learners.*

If the time taken to accept a string w in sequential learner is $t_s(w)$ and in parallel learner is $t_p(w)$, then

$$t_p(w) \leq t_s(w) \forall w$$

given both using same rule tables, and same Universal Turing Machines.

Proof. This is pretty elementary. For sequential learner acts in tandem, the time taken to accept a string is precisely the time taken to reject the string by all the other simulation previously plus the time taken to accept it. That gives:-

$$t_s(w) = \sum_{j=1}^{k-1} t_r(w, j) + t_a(w, k)$$

where the string gets accepted using the k 'th rule table, and $t_r(w, j)$ are time taken to reject the string by j th rule table, with $t_a(w, k)$ is time taken to accept the string by simulating k th rule table.

However, for the parallel case :-

$$t_p(w) = t_a(w, k)$$

Therefore,

$$t_s(w) = \sum_{j=1}^{k-1} t_r(w, j) + t_p(w)$$

which clearly establishes the theorem. \square

We should actually ask the storage complexity of the simulations. That is, to set up the sequential learner, and the parallel learner, how much storage space is needed?

We note that the storage space σ in either case is a function of the number of rule tables n , because one needs to incorporate the newly added rule table. Given the storage required to store the rule table R_k be $\sigma_r(k)$, and an Universal Turing Machine σ_t we have the precise relation as the next theorem:-

Theorem 4.3. Storage Space Comparison of the Learners.

If storage space of the sequential learner is σ_s and in parallel learner is σ_p , then

$$\sigma_s = \sigma_p$$

given both using same rule tables, and same Universal Turing Machines.

Proof. We note that:-

$$\sigma_s = \sigma_t + \sum_{k=1}^n \sigma_r(k)$$

The parallel machine needs to have $n + 1$ Turing machines, but the rule for every universal turing machine is the same. And that is never going to get changed. Then only copy of the rule table for the Universal Turing machine itself suffices. Then

$$s_p = \sigma_t + \sum_{k=1}^n \sigma_r(k)$$

which would imply that $\sigma_s = \sigma_p$. \square

Now we ask the space complexity of the simulations done both by the sequential and the parallel learner.

Theorem 4.4. *Space Complexity Comparison of the Learners.*

If the space used to accept a string w in sequential learner is $s_s(w)$ and in parallel learner is $s_p(w)$, then

$$s_s(w) \leq s_p(w) \quad \forall w$$

given both using same rule tables, and same Universal Turing Machines.

Proof. The proof is again elementary, we establish the precise relation between them. Assume that the space required to reject the string by j th simulation be $s_r(w, j)$ and to accept by k th simulation is $s_a(w, k)$. Then,

$$s_s(w) = \sup\{s_r(w, 1), \dots, s_r(w, k-1), s_a(w, k)\}$$

But,

$$s_p(w) = s_a(w, k) + \sum_{j=1, j \neq k}^n s_r(w, j)$$

which immediately establishes the theorem. \square

Theorem 4.5. *Learning Complexity Comparison of the Learners.*

Let the expected time taken to learn a new language class C_N by sequential learner be $\tau_s(C_N)$ and that of a parallel learner be $\tau_p(C_N)$. Then,

$$\tau_p(C_N) \leq \tau_s(C_N)$$

Proof. Given the language class C_N can actually be incorporated, that is C_N has no string for which the sequential learner would go into infinite loop, the learning actually requires cloning and a lucky mutation for both type of the systems. So, in which case the time taken is the same as expected time for the lucky mutation τ_μ . In which case :-

$$\tau_p(C_N) = \tau_s(C_N) = \tau_\mu$$

But, when $w \in C_N$ makes the sequential system get into infinite loop, then $\tau_s(C_N) \rightarrow \infty$, and generally:-

$$\tau_p(C_N) \leq \tau_s(C_N)$$

holds, as stated. \square

These theorems clearly establishes that when space is not premium, then, parallel learner would be a more optimal strategy for learning. In specificity, if accuracy and completeness is needed, then parallel learners are better then the sequential ones.

But that is not all. The crux of this lies in the fact that autonomous learning in nature has to be inherently blind, triggered by chance mutations. Given that, there would be no way to ensure an *out of the system decision making* to further optimize the design of the resulting system.

Due to the nature of the bling learning, even if optimisation is possible, there has to be parallelism to complete the learner. This *intelligent* ordering can evolve in nature, and is discussed in the next theorem.

Theorem 4.6. *Existence of a Hybrid Learning System*

There exists a hybrid model of learning which uses parallelism and serial model, and is complete.

Proof. We prove it by constructing a hybrid system. We note that the halting problem establishes a partial order relation over the rule tables. It can be stated as :-

$R_i \leq R_j$ iff $w_j \in C(R_j)$ hangs the simulation of R_i .

Given this partial order relation we can create sets \mathcal{U}_k (for *unrelated*) so that :-

$$R_a \in \mathcal{U}_k \text{ iff } \nexists R_b \in \mathcal{U}_k \text{ s.t. } R_a \leq R_b \text{ or } R_b \leq R_a$$

Let $U = \{\mathcal{U}_k\}$. Now individual rules $R_{ki} \in \mathcal{U}_k$ can be run in sequential. But cross set rules $R_{ki} \in \mathcal{U}_k$ and $R_{pi} \in \mathcal{U}_p$ can not be run in sequential.

So, if a system is capable of running the $n = |U|$ parallel execution, we can make a complete hybrid system that runs *related* rules parallel, but *unrelated* rules can be run sequentially.

This completes the proof. □

We note that construction of such a system is not algorithmic, that is not computationally possible. The step of finding out the relations between language class rules is not computable. Hence, only blind mutation can create such a system, that is by chance. However, up until this section we have established that parallelism, in general, is inherent in nature, even if a Hybrid system is evolved, there is parallelism inherent in it. In the next section we show that why parallel learning strategy dominates nature.

5. PARALLEL STRATEGIES AND EVOLUTION

In this section we finally ask the following question :

Given the sequential and parallel strategies available, which strategy would dominate in nature?

This obviously should depend upon the concept of which strategy *pays off more* for survival. But payoffs like this are in the realm of *Game Theory* [21]. In the context of the Game Theory payoffs are represented as numbers which represent the motivations of players. Payoffs may represent profit, quantity, that is any “utility”.

As resources are limited in nature, gaining advantage or loosing advantage can be modelled by a *fixed sum game*[21] where a both the players are competing for a fixed sum in reward. However, we can generalise any *fixed sum game* into a *zero sum game* by setting the fixed amount at 0 [21][17][18].

Given *accepting all strings having the same utility*, we can have the payoffs *proportional to the cardinality of the language class accepted by the strategy*. In general, if learner p_1 plays with S_1 learning strategy, and learner p_2 plays with S_2 learning strategy, then, intuitively, the player p_1 is in advantage if $|C(S_1)| > |C(S_2)|$, and for p_2 it is vice versa with $|X|$ denoting the cardinality of set X . Hence, payoff of S_1 against S_2 denoted by $E(S_1, S_2)$ can be calculated as:-

$$E(S_1, S_2) = |C(S_1)| - |C(S_2)|$$

When $C(S)$ becomes infinite, to define the payoff formally we have to resort to the *measure theory* [19].

Definition 5.1. Utility Measure.

Let $A, B, X, Y \subset \Sigma^+$ be a set of strings. A measure $\mathcal{U} : \Sigma^+ \rightarrow \mathbb{R}_+$ is a utility measure, iff:-

$$\mathcal{U}(X) = 0 \text{ iff } X = \emptyset$$

implying:-

$$A \subset B \implies \mathcal{U}(B) > \mathcal{U}(A)$$

where $\mathcal{U}(X) > \mathcal{U}(Y)$ implies that the set X has more utility than Y .

Note that with this measure in place, we do not need any assumptions about the utilities of different strings. It also glorifies the age old saying: “*no learning is useless*”, only this time, more formally.

Definition 5.2. Payoff between Learning Strategies.

Let $S_1, S_2 \in \mathcal{S}$ are two learning strategies. Let $C(S_i)$ denotes the language class accepted by the strategy S_i . Let $\bigcup C(S_k) = C(\mathcal{S})$. Let \mathcal{U} be a utility measure (5.1) defined over $C(\mathcal{S})$. Then, the payoff of strategy S_1 against S_2 is given by:-

$$E(S_1, S_2) = \mathcal{U}(C(S_1)) - \mathcal{U}(C(S_2))$$

In particular, if $S_1 = S_2 = S$ then,

$$E(S, S) = 0$$

Now we make the bold claim that *parallel strategy is evolutionarily stable*. To do so we need to state the definition of an *evolutionarily stable strategy* [16], [17],[21] [18].

Definition 5.3. Evolutionarily stable strategy (ESS). Let \mathcal{S} is a set of strategies. Let $E(S, T)$ represent the payoff for playing strategy S against strategy T . The strategy S is ESS iff one of the following conditions holds $\forall T \neq S$ with $S, T \in \mathcal{S}$:-

(1) *Strict Nash Equilibrium :*

$$\forall T \in \mathcal{S} ; E(S, S) > E(T, S)$$

(2) *Maynard Smith's second Condition:*

$$\forall T \in \mathcal{S} ; E(S, S) = E(T, S) \text{ and } E(S, T) > E(T, T)$$

Now with this definition in hand, we establish the criterion for ESS in the evolution of learning.

Theorem 5.1. *ESS for Learning Strategies.*

Let \mathcal{S} be a set of learning strategies. Let $S_e \in \mathcal{S}$ be a learning strategy with:-

$$\forall S_k \in \mathcal{S} ; C(S_k) \subset C(S_e)$$

where $C(S)$ is language class accepted by strategy S . Then, the strategy S_e is ESS.

Proof. We note that :-

$$E(S_e, S_k) = \mathcal{U}(C(S_e) \cup C_E) - \mathcal{U}(C(S_k))$$

where $C(S_e) \cap C_E = \emptyset$, which implies :-

$$\forall S_k \in \mathcal{S} ; E(S_e, S_k) = \mathcal{U}(C_E) \implies E(S_e, S_k) > 0$$

implying :-

$$\forall S_k \in \mathcal{S} ; E(S_k, S_e) = -\mathcal{U}(C_E) \implies E(S_k, S_e) < 0$$

By definition we have $E(S_e, S_e) = 0$ and therefore,

$$\forall S_k \in \mathcal{S} ; E(S_e, S_e) > E(S_k, S_e)$$

Comparing from definition (5.3, rule 1), S_e is an ESS. In fact we note that this ESS is a *strict Nash Equilibrium*. \square

Now we establish that the parallel strategies are ESS.

Theorem 5.2. *Parallel learning strategies are ESS.*

Let \mathcal{S} be a set of strategy such that $\forall S \in \mathcal{S}$ has rule table sets $R(S) = \mathcal{R}$. Let $S_p \in \mathcal{S}$ is a parallel strategy. S_p is an ESS.

Proof. We note that the language class accepted by the tandem learners are a strict subset of the language class accepted by the parallel learner by the theorems (3.1,3.3,4.1). That is, if C_t is language class for the sequential learners, and C_p is the language class of parallel learners,

$$C_t \subset C_p$$

Now using the theorem 5.1 we can immediately deduce that parallel strategies are ESS. \square

6. SUMMARY AND FUTURE WORKS

It is really surprising that the parallel execution model which is notorious for creating issues in everyday computing, is in fact, should be prevalent in nature. In this paper we have demonstrated why in nature parallel strategies are the optimally suited one. Once nature gets invaded by the parallel strategies, there is no going back, because parallel strategies are ESS. We established this fact using Church Turing Thesis, and an utility measure which tallies with common sense. This

demonstrates the power of computer science as a proper science, fully capable of describing natural phenomenon, outside the realm of rather artificial settings.

Although we have established why nature prefers parallel execution, we did not establish a typical learning time for any language class, and the expected time to evolve into a hybrid learning system. These studies would be done in the future.

REFERENCES

- [1] BOUCHER, ANDREW. *Parallel Machines* . Minds and Machines Volume 7 : 1997 , 543-551
- [2] COHEN, I. A. DANIEL , *Introduction to Computer Theory* . Wiley; 2 edition, 1996.
- [3] AHO; MOTWANI ; ULLMAN , *Introduction to Automata Theory, Languages, and Computation* . Prentice Hall; 3 edition (July 9, 2006).
- [4] SIPSER, MICHAEL , *Introduction to the Theory of Computation* . Course Technology; 3 edition (June 27, 2012).
- [5] SEARLE, JOHN. , *Is the Brain's Mind a Computer Program?* . Scientific American, Jan. 1990, pp. 20-25.
- [6] TURING, ALAN M. , *On Computable Numbers, With an Application to the Entscheidungsproblem* . Proc. London Math. Soc. Ser. 2 42, 1936 ,pp. 230-265.
- [7] TURING, ALAN M. , *Computing Machinery and Intelligence* . Mind 59, 1950 , pp. 433-460.
- [8] WIEDERMAN, JURAJ. , *Parallel Turing Machines* . 1984
- [9] ROSENBLATT, F. , *The Perceptron: A Probabilistic Model For Information Storage And Organization In The Brain* . Psychological Review 65 (6), 1958 , pp. 386-408.
- [10] RUMELHART, D.E ; MCCLELLAND, JAMES. , *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Cambridge: MIT Press, 1986
- [11] HOFSTADTER , DOUGLAS R. , *Gödel, Escher, Bach: An Eternal Golden Braid* . Basic Books; 20 Anv edition (February 5, 1999).
- [12] BEN-JACOB E. , *Learning from bacteria about natural information processing* . Ann N Y Acad Sci. 2009 Oct;1178:78-90. doi: 10.1111/j.1749-6632.2009.05022.x.
- [13] RAMACHANDRAN , V.S. , *Phantoms in the Brain* . Harpercollins Pb; New Ed edition (May 20, 1999).
- [14] PENROSE , ROGER. , *Emperor's New Mind* . Random House UK (February 27, 1994).
- [15] PENROSE , ROGER. , *Shadows of the Mind: A Search for the Missing Science of Consciousness* . Oxford University Press, USA; Reprint edition (August 22, 1996).
- [16] DAWKINS , RICHARD. , *Selfish Gene* . Scientific American (2004)
- [17] DAWKINS , RICHARD. , *Climbing Mount Improbable* . W. W. Norton & Company (September 17, 1997).
- [18] SMITH, MAYNARD ; J. PRICE, G.R , *The Logic of Animal Conflict*. Nature 246, (02 November 1973), pp. 15-18.
- [19] DOOB , J.L. , *Measure Theory* . Springer (1993).
- [20] CHUNG , K.L. ; FARID, A. , *Elementary Probability Theory with Stochastic Processes and an Introduction to Mathematical Finance* . Springer 4th edition (2003).
- [21] BARRON , E.N. , *Game Theory: An Introduction* . Wiley India Pvt Ltd (2009).

D.E.SHAW & CO. INDIA, HYDERABAD
E-mail address: mondal@deshaw.com

MICROSOFT INDIA, HYDERABAD
E-mail address: parthag@microsoft.com